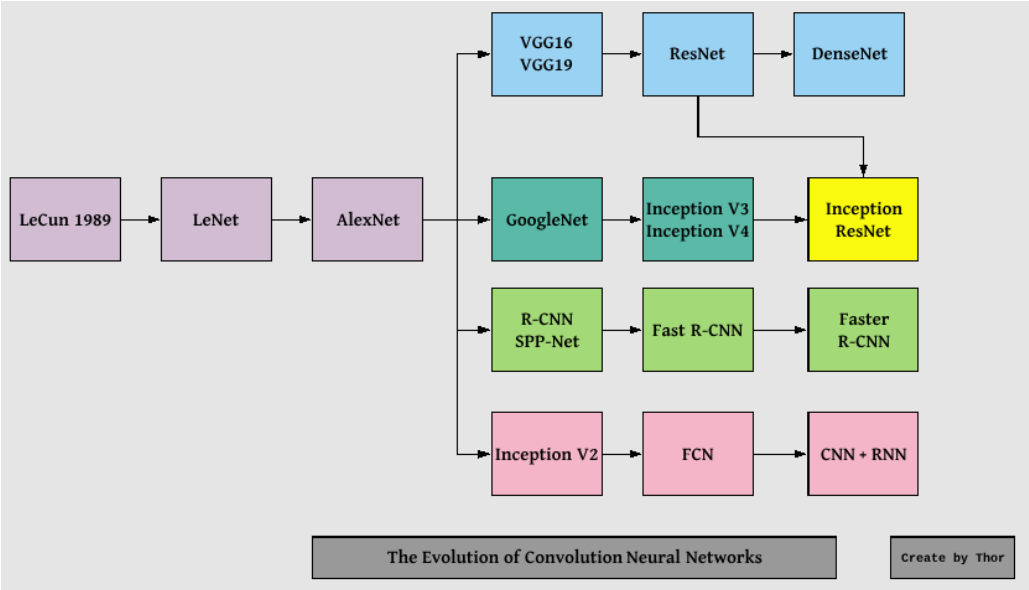
Quá trình phát triển của CNN từ LeNet đến DenseNet.

DEEP LEARNING VÀ ỨNG DỤNG · SUNDAY, JULY 8, 2018

Mở đầu

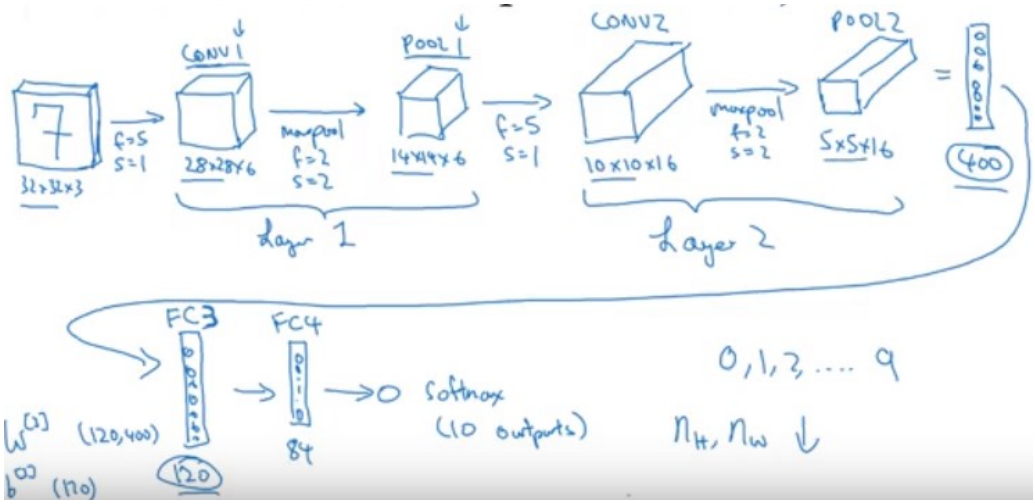
Convolutional neural network là một mạng neural được ứng dụng rất nhiều trong deep learning trong computer vision cho classifier và localizer . Từ mạng CNN cơ bản người ta có thể tạo ra rất nhiều architect khác nhau, từ những mạng neural cơ bản 1 đến 2 layer đến 100 layer. Đã bao giờ bạn tự hỏi nên sử dụng bao nhiêu layer, nên kết hợp conv với maxpooling thế nào? conv-maxpooling hay conv-conv-maxplooling ? hay nên sử dụng kernel 3x3 hay 5x5 thậm chí 7x7 điểm khác biệt là gì ? Làm gì khi model bị vanishing/exploding gradient, hay tại sao thi thêm nhiều layer hơn thì theo lý thuyết accuarcy phải cao hơn so với shallow model, nhưng thực tế lại không phải accuarcy không tăng thậm chí là giảm đó có phải nguyên nhân do overfitting .Trong bài viết này ta sẽ tìm hiểu các architure nổi tiếng để xem cấu trúc của nó như thế nào, các ý tưởng về CNN mới nhất hiện nay từ đó ta có thể trả lời được mấy câu hỏi trên.



Quá trình phát triển của CNN

1.LeNet(1998).

LeNet là một trong những mạng CNN lâu đời nổi tiếng nhất được Yann LeCUn phát triển vào những năm 1998s. Cấu trúc của LeNet gồm 2 layer (Convolution + maxpooling) và 2 layer fully connected layer và output là softmax layer .



LeNet (Source CNN của Andrew Ng)

Chúng ta cùng tìm hiểu chi tiết architect của LeNet đối với dữ liệu mnist (accuracy lên đến 99%).

- Input shape 28x28x3
- Layer 1 :
  - Convolution layer 1 : Kernel 5x5x3 , stride = 1,no padding, number filter = 6 ,output = 28x28x6.
  - Maxpooling layer : pooling size 2x2,stride = 2,padding = “same”,output = 14x14x6.
- Layer 2 :
  - Convolution layer 2 : kernel 5x5x6,stride = 1, no padding, number filter = 16,output = 10x10x16.
  - Maxpooling layer : pooling size = 2x2, stride = 2, padding =”same”,output = 5x5x16.
- Flatten output = 5x5x16 = 400
- Fully connection 1 : output = 120
- Fully connection 2 : output = 84
- Softmax layer, output = 10 (10 digits).

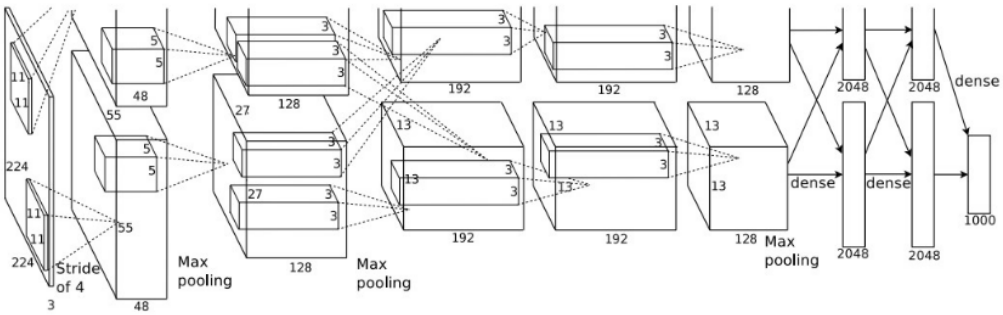
Nhược điểm của LeNet là mạng còn rất đơn giản và sử dụng sigmoid (or tanh) ở mỗi convolution layer mạng tính toán rất chậm.

2. Alexnet(2012).

AlexNet là một mạng CNN đã dành chiến thắng trong cuộc thi ImageNet LSVRC-2012 năm 2012 với large margin (15.3% VS 26.2% error rates). AlexNet là một mạng CNN traning với một số lượng parameter rất lớn (60 million) so với LeNet. Một số đặc điểm:

- Sử dụng relu thay cho sigmoid(or tanh) để xử lý với non-linearity. Tăng tốc độ tính toán lên 6 lần.

- Sử dụng dropout như một phương pháp regularization mới cho CNN. Dropout không những giúp mô hình tránh được overfitting mà còn làm giảm thời gian huấn luyện mô hình
- Overlap pooling để giảm size của network (Traditionally pooling regions không overlap).
- Sử dụng local response normalization để chuẩn hóa ở mỗi layer.
- Sử dụng kỹ thuật data augmentation để tạo thêm data training bằng cách translations, horizontal reflections.
- Alexnet training với 90 epochs trong 5 đến 6 ngày với 2 GTX 580 GPUs. Sử dụng SGD với learning rate 0.01, momentum 0.9 và weight decay 0.0005.



AlexNet (Nguồn ImageNet Classification with Deep Convolutional Neural Networks)

Architect của Alexnet gồm 5 convolutional layer và 3 fully connected layer. Activation Relu được sử dụng sau mỗi convolution và fully connection layer. Detail architecture với dataset là imagenet size là 227x227x3 với 1000 class (khác với trong hình trên size là 224x224).

Detail Architect:

- Input shape 227x227x3.
- Layer 1 :
  - Conv 1 : kernel : 11x11x3,stride = 4,no padding, number = 96,activation = relu,output = 55x55x96.
  - Maxpooling layer : pooling size = 3x3,stride = 2,padding =”same” ,output = 27x27x96.
  - Normalize layer.
- Layer 2 :
  - Conv 2 : kernel :3x3x96,stride = 1, padding = “same”, number filter = 256,activation = relu,output = 27x27x256.
  - Maxpooling layer : pooling size = 3x3,stride=2, padding =”same”,output = 13x13x256.
  - Normalize layer.
- Layer 3:

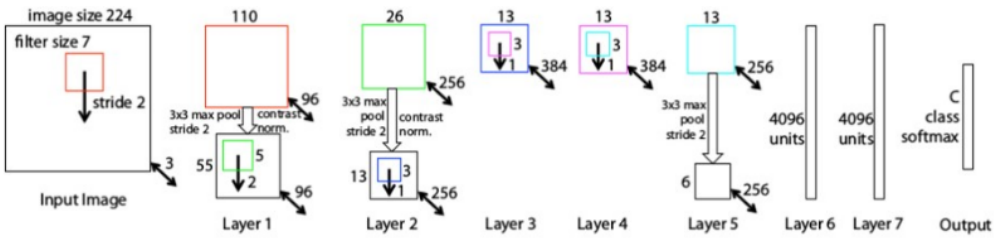
- Conv 3 : kernel :3x3x256, stride = 1,padding=”same”, number filter = 384, activation = relu, output = 13x13x384.

- Layer 4:
  - Conv 4 : kernel : 3x3x384 , stride = 1, padding = “same”, number filter = 384, activation= relu, output = 13x13x384
- Layer 5 :
  - Conv 5 : kernel 3x3x384, stride = 1, padding = “same”, number filter = 256, activation = relu, output = 13x13x256.
  - Pooling layer : pooling size = 3x3,stride =2,padding =”same”,output = 6x6x256.
- Flatten 256x6x6 = 9216
- Fully connection layer 1 : activation = relu , output = 4096 + dropout(0.5).
- Fully connection layer 2 : activation = relu , output = 4096 + dropout(0.5).
- Fully connection layer 3 : activation = softmax , output = 1000 (number class)

3. ZFNet(2013).

ZFNet là một mạng CNN winner trong ILSVRC 2013 với top-5 error rate của 14.8% . ZFNet có cấu trúc rất giống với AlexNet với 5 layer convolution , 2 fully connected layer và 1 output softmax layer. Khác biệt ở chỗ kernel size ở mỗi Conv layer .Một số đặc điểm chính :

- Tương tự AlexNet nhưng có một số điều chỉnh nhỏ.
- Alexnet training trên 15m image trong khi ZF training chỉ có 1.3m image.
- Sử dụng kernel 7x7 ở first layer (alexnet 11x11).Lý do là sử dụng kernel nhỏ hơn để giữ lại nhiều thông tin trên image hơn.
- Tăng số lượng filter nhiều hơn so với AlexNet
- Training trên GTX 580 GPU trong 20 ngày



ZFNet (Source Visualizing and Understanding Convolutional Networks)

Detail architect:

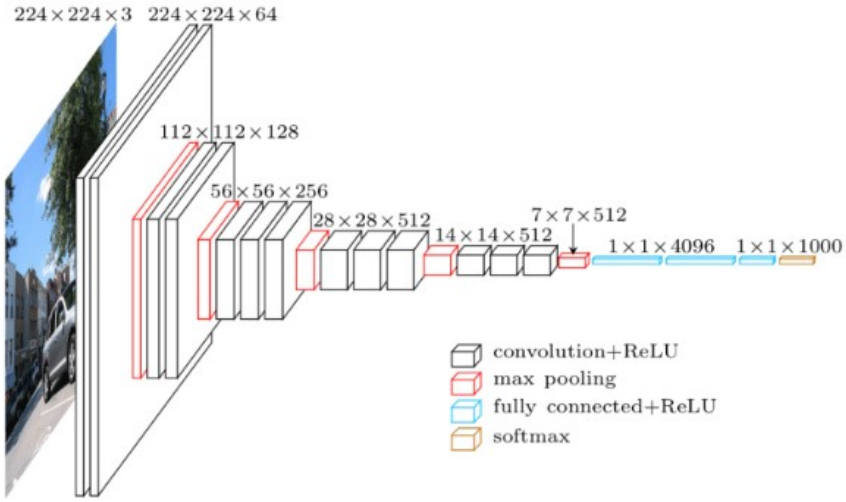
- Input shape 224x224x3 .
- Layer 1 :

- Conv 1 : kernel = 7x7x3, stride = 2, no padding, number filter = 96, output = 110x110x96.
- Maxpooling 1 : pooling size = 3x3,stride=2, padding = “same”,output = 55x55x96
- Normalize layer.

- Layer 2 :
  - Conv 2 : kernel = 5x5x96, stride = 2, no padding, number filter = 256, output = 26x26x256.
  - Maxpooling 2 : pooling size = 3x3, stride=2, padding = “same”,output = 13x13x256
  - Normalize layer.
- Layer 3:
  - Conv 3 : kernel = 3x3x256, stride=1, padding=”same”, number filter = 384,output = 13x13x384.
- Layer 4 :
  - Conv 4 : kernel = 3x3x384, stride=1, padding=”same”, number filter = 384,output = 13x13x384.
- Layer 5 :
  - Conv 5 : kernel = 3x3x384, stride=1, padding=”same”, number filter = 256,output = 13x13x256.
  - Maxpooling : pooling size = 3x3,stride =2,padding =”same”,output = 6x6x256.
- Flatten 6x6x256 = 9216
- Fully connection 1 : activation = relu,output =4096
- Fully connection 2 : activation = relu,output =4096
- Softmax layer for classifier ouput = 1000

4. VGGNet(2014).

Sau AlexNet thì VGG ra đời với một số cải thiện hơn , trước tiên là model VGG sẽ deeper hơn, tiếp theo là thay đổi trong thứ tự conv. Từ LeNet đến AlexNet đều sử dụng Conv-maxpooling còn VGG thì sử dụng 1 chuỗi Conv liên tiếp Conv-Conv-Conv ở middle và end của architect VGG. Việc này sẽ làm cho việc tính toán trở nên lâu hơn nhưng những feature sẽ vẫn được giữ lại nhiều hơn so với việc sử dụng maxpooling sau mỗi Conv. Hơn nữa hiện nay với sự ra đời của GPU giúp tốc độ tính toán trở nên nhanh hơn rất nhiều lần thì vấn đề này không còn đáng lo ngại. VGG cho small error hơn AlexNet trong ImageNet Large Scale Visual Recognition Challenge (ILSVRC) năm 2014. VGG có 2 phiên bản là VGG16 và VGG19.



VGG16 (source internet)

Architect của VGG16 bao gồm 16 layer : 13 layer Conv (2 layer conv-conv,3 layer conv-conv-conv) đều có kernel 3x3, sau mỗi layer conv là maxpooling downsize xuống 0.5, và 3 layer fully connection. VGG19 tương tự như VGG16 nhưng có thêm 3 layer convolution ở 3 layer conv cuối (thành 4 conv stack với nhau).

Detail parameter VGG16 :

```
INPUT: [224x224x3]        memory:  224*224*3=150K   weights: 0
CONV3-64: [224x224x64]  memory:  224*224*64=3.2M   weights: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64]  memory:  224*224*64=3.2M   weights: (3*3*64)*64 = 36,864
POOL2: [112x112x64]  memory:  112*112*64=800K   weights: 0
CONV3-128: [112x112x128]  memory:  112*112*128=1.6M   weights: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128]  memory:  112*112*128=1.6M   weights: (3*3*128)*128 = 147,456
POOL2: [56x56x128]  memory:  56*56*128=400K   weights: 0
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*256)*256 = 589,824
POOL2: [28x28x256]  memory:  28*28*256=200K   weights: 0
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512]  memory:  14*14*512=100K   weights: 0
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512]  memory:  7*7*512=25K  weights: 0
FC: [1x1x4096]  memory:  4096  weights: 7*7*512*4096 = 102,760,448
FC: [1x1x4096]  memory:  4096  weights: 4096*4096 = 16,777,216
FC: [1x1x1000]  memory:  1000 weights: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters
```

VGG16 parameter (source cs231)

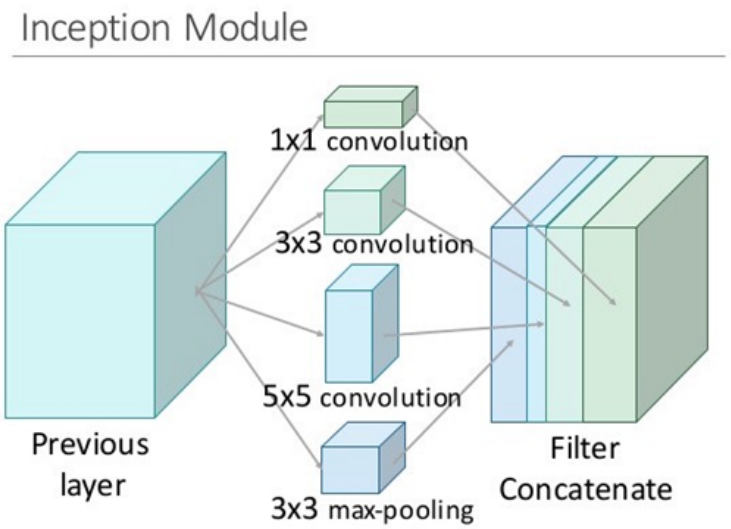
Một số đặc điểm :

- Sử dụng kernel 3x3 thay vì 11x11 ở alexnet(7x7 ZFNet). Kết hợp 2 conv 3x3 có hiểu quả hơn 1 cov 5x5 về receptive field giúp mạng deeper hơn lại giảm tham số tính toán cho model.
- 3 Conv 3x3 có receptive field same 1 conv 7x7.

- Input size giảm dần qua các conv nhưng tăng số chiều sâu.
- Làm việc rất tốt cho task classifier và localizer (rất hay được sử dụng trong object detection).
- Sử dụng relu sau mỗi conv và training bằng batch gradient descent.
- Có sử dụng data augmentation technique trong quá trình training.
- Training với 4 Nvidia Titan Black GPUs trong 2-3 tuần.

5. GoogleNet(2014).

Năm 2014, google publish một mạng neural do nhóm research của họ phát triển có tên là googleNet. Nó performance tốt hơn VGG, googleNet 6.7% error rate trong khi VGG là 7.3% Ý tưởng chính là họ tạo ra một module mới có tên là inception giúp mạng traning sâu và nhanh hơn, chỉ có 5m tham số so với alexnet là 60m nhanh hơn gấp 12 lần. Inception module là một mạng CNN giúp training wider(thay vì thêm nhiều layer hơn vì rất dễ xảy ra overfitting + tăng parameter người ta nghĩ ra tăng deeper ở mỗi tầng layer) so với mạng CNN bình thường. Mỗi layer trong CNN truyền thống sẽ extract các thông tin khác nhau. Output của 5x5 conv kernel sẽ khác với 3x3 kernel. Vậy để lấy những thông tin cần thiết cho bài toán của chúng ta thì nên dùng kernel size như thế nào ? Tại sao chúng sử dụng tất cả ta và sau đó để model tự chọn. Đó chính là ý tưởng của Inception module, nó tính toán các kernel size khác nhau từ một input sau đó concatenate nó lại thành output.



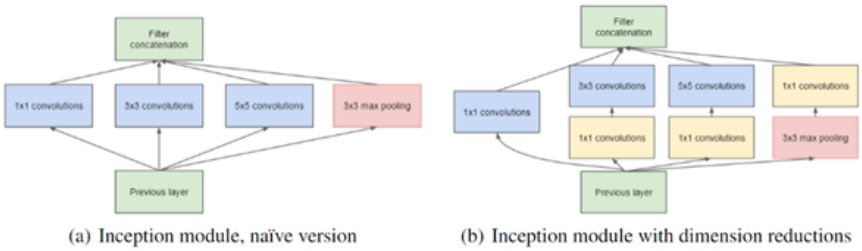
Trong inception người ta dùng conv kernel 1x1 với 2 mục đích là giảm tham số tính toán và dimensionality reduction . Dimensionality reduction có thể hiểu làm giảm depth của input (vd iput 28x28x100 qua kernel 1x1 với filter = 10 sẽ giảm depth về còn 28x28x10). Giảm chi phí tính toán có thể hiểu qua ví dụ sau :

- Input shape 28x28x192 qua kernel 5x5 với 32 thì ouput là 28x28x32(padding same) thì tham số tính toán là (5x5x192)*(28x28x32)=120 million

- Input shape 28x28x192 qua kernel 1x1x192 filter = 16 , output = 28x28x16 tiếp tục với kernel 5x5x16 filter = 32 đươch output = 28x28x32. Tổng tham số tính toán : (28x28x16)*192 + (28x28x32)*(5x5x16) = 2.4 + 10 = 12.4 million. Ta thấy với cùng output là 28x28x32 thì nếu dùng kernel 5x5x192 với 32 filter thì sẽ có tham số gấp 10 lần so với sử dụng kernel 1x1x192 sau đó dùng tiếp 1 kernel 5x5x16 với filter 32.

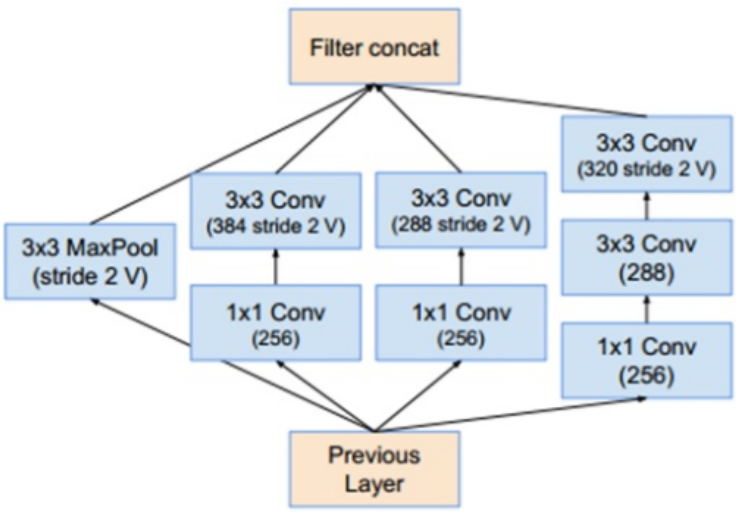
Inception hiện giờ có 4 version , ta sẽ cùng tìm hiểu sơ qua các version:

- Inception v1 : có 2 dạng là naïve và dimension reduction. Khác biệt chính đó là version dimension reduction nó dùng conv 1x1 ở mỗi layer để giảm depth của input giúp model có ít tham số hơn. Inception naïve có architect gồm 1x1 conv,3x3 conv, 5x5 conv và 3x3 maxpooling.



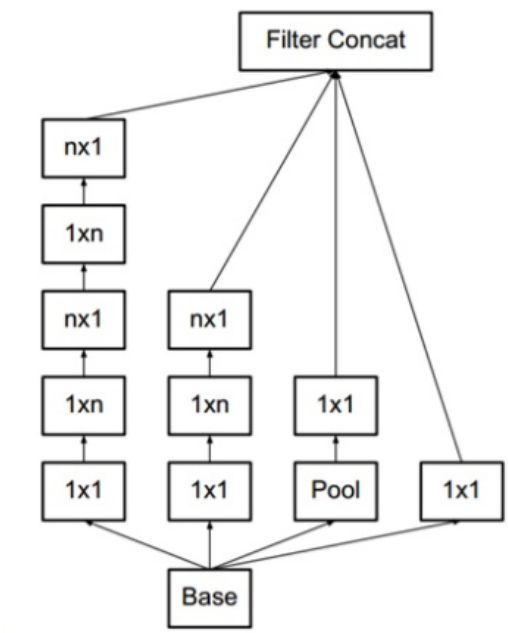
Inception v1 (source internet)

- Inception v2 : Cải thiện version 1, thêm layer batchnormalize và giảm Internal Covariate Shift. Ouput của mỗi layer sẽ được normalize về Gaussian N(0,1). Conv 5x5 sẽ được thay thế bằng 2 conv 3x3 để giảm computation cost.



Inception v2 (source v2)

- Inception v3 : Điểm đáng chú ý ở version này là Factorization. Conv 7x7 sẽ được giảm về conv 1 dimesion là (1x7),(7x1). Tương tự conv 3x3 (3x1,1x3). Tăng tốc độ tính toán. Khi tách ra 2 conv thì làm model deeper hơn.



Inception v3 (source internet)

Inception v4 : là sự kết hợp inception và resnet. Detail googleNet architect :

Detail Architect googlenet(v1):

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

GoogleNet (source internet)

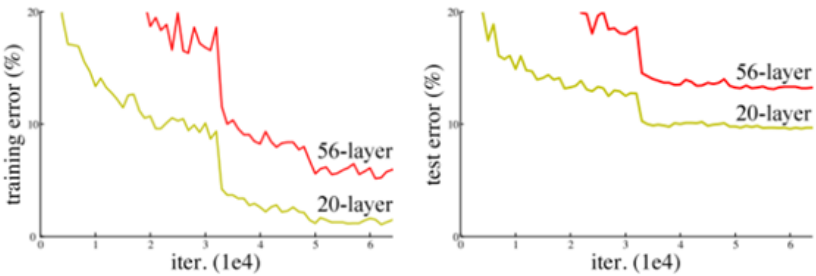
Một số đặc điểm :

- GoogleNet gồm 22 layer, khởi đầu vẫn là những simple convolution layer, tiếp theo là những block của inception module với maxpooling theo sau mỗi block. Một số đặc điểm chính.
- Sử dụng 9 Inception module trên toàn bộ architect. Làm model deeper hơn rất nhiều.
- Không sử dụng fully connection layer mà thay vào đó là average pooling từ 7x7x1024 volume thành 1x1x1024 volume giảm thiểu được rất nhiều parameter.

- Ít hơn 12x parameter so với Alexnet.
- Auxiliary Loss được add vào total loss(weight =0.3). Nhưng được loại bỏ khi test.

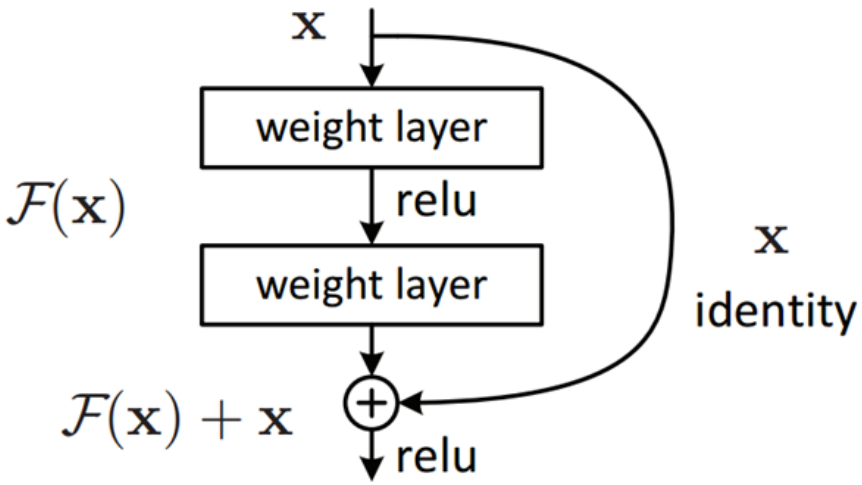
6. ResNets(2015).

ResNet được phát triển bởi microsoft năm 2015 với paper " Deep residual learning for image recognition". ResNet winer ImageNet ILSVRC competition 2015 với error rate 3.57% ,ResNet có cấu trúc gần giống VGG với nhiều stack layer làm cho model deeper hơn. Không giống VGG, resNet có depth sâu hơn như 34,55,101 và 151 . Resnet giải quyết được vấn đề của deep learning truyền thống , nó có thể dễ dàng training model với hàng trăm layer. Để hiểu ResNet chúng ta cần hiểu vấn đề khi stack nhiều layer khi training, vấn đề đầu tiên khi tăng model deeper hơn gradient sẽ bị vanishing/explodes. Vấn đề này có thể giải quyết bằng cách thêm Batch Normalization nó giúp normalize output giúp các hệ số trở nên cân bằng hơn không quá nhỏ hoặc quá lớn nên sẽ giúp model dễ hội tụ hơn. Vấn đề thứ 2 là degradation, Khi model deeper accuracy bắt đầu bão hòa(saturated) thậm chí là giảm. Như hình vẽ bên dưới khi stack nhiều layer hơn thì training error lại cao hơn ít layer như vậy vấn đề không phải là do overfitting. Vấn đề này là do model không dễ training khó học hơn, thử tượng tượng một training một shallow model, sau đó chúng ta stack thêm nhiều layer , các layer sau khi thêm vào sẽ không học thêm được gì cả (identity mapping) nên accuracy sẽ tương tự như shallow model mà không tăng. Resnet được ra đời để giải quyết vấn đề degradation này.



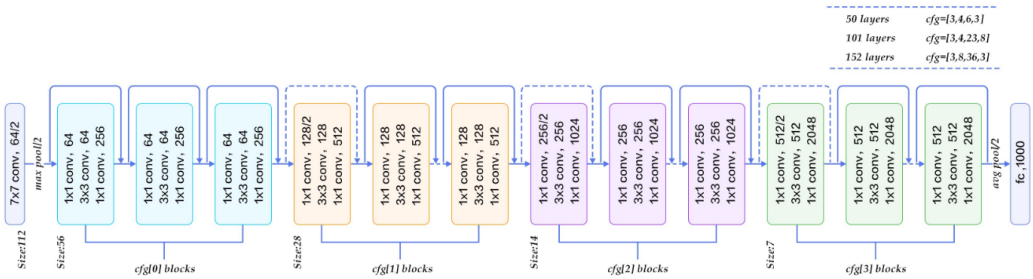
Deep Residual Learning for Image Recognition

ResNet có architect gồm nhiều residual block, ý tưởng chính là skip layer bằng cách add connection với layer trước. Residual block là feed foword x(input) qua một số layer conv-max-conv, ta thu được F(x) sau đó add thêm x vào H(x) = F(x) + x . Model sẽ dễ học hơn khi chúng ta thêm feature từ layer trước vào.



Resnet block (source internet)

Detail Architect :



ResNet (source Internet)

Detail parameter :

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Parameter Resnet (source Deep Residual Learning for Image Recognition)

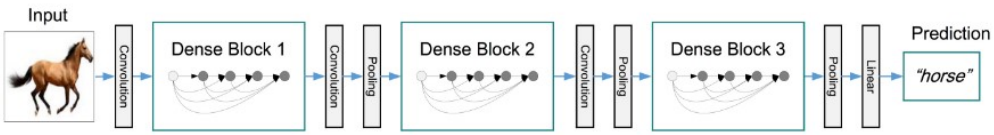
Một số đặc điểm :

- Sử dụng batch Normalization sau mỗi Conv layer.
- Initialization Xavier/2

- Training với SGD + momentum(0.9)
- Learning rate 0.1, giảm 10 lần nếu error ko giảm
- Mini batch size 256
- Weight decay 10^-5
- Không sử dụng dropout

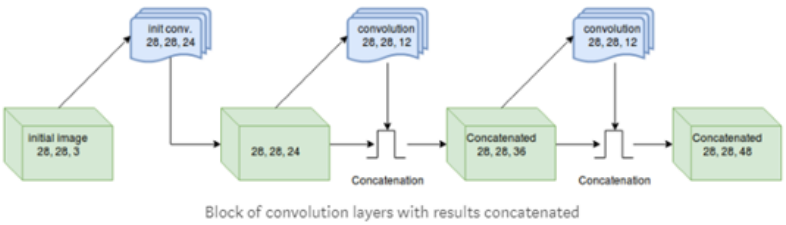
7. Densenet(2016)

Densenet(Dense connected convolutional network) là một trong những netwok mới nhất cho visual object recognition. Nó cũng gần giống Resnet nhưng có một vài điểm khác biệt. Densenet có cấu trúc gồm các dense block và các transition layers. Được stack dense block-transition layers-dense block- transition layers như hình vẽ. Với CNN truyền thống nếu chúng ta có L layer thì sẽ có L connection, còn trong densenet sẽ có L(L+1)/2 connection.



Densely Connected Convolutional Networks

Hãy tưởng tượng ban đầu ta có 1 image size (28,28,3). Đầu tiên ta khởi tạo feature layer bằng Conv tạo ra 1 layer size (28,28,24). Sau mỗi layer tiếp theo (Trong dense block) nó sẽ tạo thêm K= 12 feature giữa nguyên width và height. Khi đó output tiếp theo sẽ là (28,28,24 +12),(28,28,24 +12+12). Ở mỗi dense block sẽ có normalization, nonlinearity và dropout. Để giảm size và depth của feature thì transition layer được đặt giữa các dense block, nó gồm Conv kernel size =1, average pooling (2x2) với stride = 2 nó sẽ giảm output thành (14,14,48)



DenseNet (source internet)

Detail parameter :

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv, 3 × 3 conv] × 24	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 48	[1 × 1 conv, 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv, 3 × 3 conv] × 16	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Parameter DenseNet (Densely Connected Convolutional Networks)

Một số ưu điểm của Densenet:

- Accuracy : Densenet training tham số ít hơn 1 nửa so với Resnet nhưng có same accuracy so trên ImageNet classification dataset.
- Overfitting : DenseNet resistance overfitting rất hiệu quả.
- Giảm được vashing gradient.
- Sử dụng lại feature hiệu quả hơn.

Kết bài : Trên đây chỉ là phần tóm lược sơ qua các architect nổi tiếng của CNN. Bên trong đó còn có nhiều cấu trúc phức tạp vì thời gian cũng như kiến thức có hạn nên vẫn chưa viết sâu hết được. Bên cạnh đó còn có nhiều sai sót rất mong bạn đọc góp ý để mình có thể hoàn thiện bài viết hơn.

Tham khảo :

- Deep learning course : Andrew Ng
- An Intuitive Guide to Deep Network Architectures
- A Simple Guide to the Versions of the Inception Network
- CS231n: Convolutional Neural Networks for Visual Recognition
- Paper : All paper about lenet,alexnet,vgg,googlenet,resnet,densenet
- Notes on the Implementation of DenseNet in TensorFlow.
- The Efficiency of Densenet